# 样卷解析

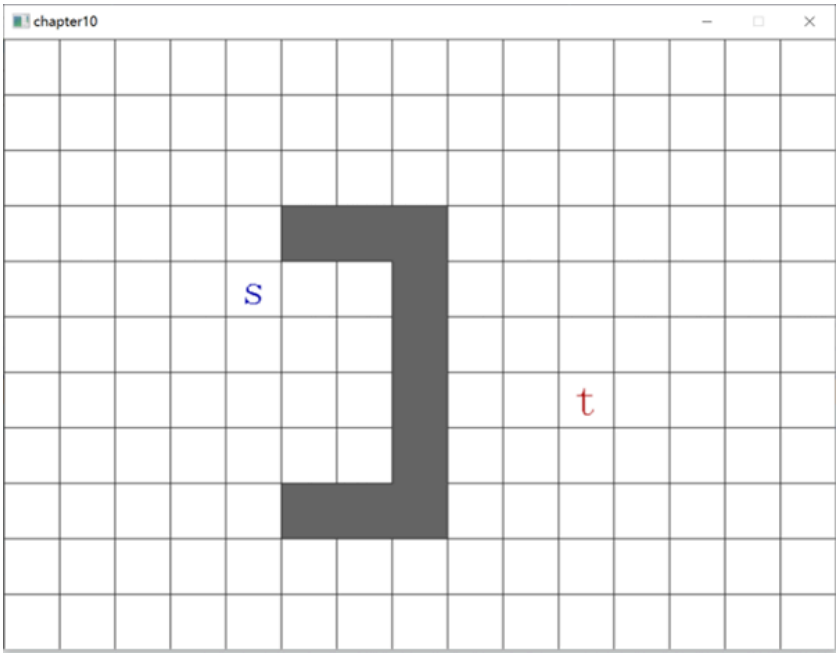
# 《算法设计与分析》考试样卷

**（考试对象：人工智能、智能科学与技术专业）**

**学号　　姓名　　成绩**

**一、算法设计与分析 （每题20分，共60分）**

## 1.写出A*算法的伪代码**；并写出在下图中，利用A*算法找到从s点到t点最短路径算法思想的示意图（白色方格可以通过，黑色方格为障碍物）。




（1）伪代码

```
算法：A-Star (A*) 路径规划
输入：起点 s，终点 t，网格地图（包含障碍物信息）
输出：从 s 到 t 的最短路径

1. 初始化：
   - 创建一个优先队列 OpenList，用于存储待考察的节点（按 f 值从小到大排序）
   - 创建一个集合 ClosedList，用于存储已考察的节点
   - 将起点 s 加入 OpenList，设置 g(s) = 0, f(s) = h(s)

2. 循环搜索（当 OpenList 不为空时）：
   a. 从 OpenList 中弹出 f 值最小的节点，设为当前节点 current
   b. 如果 current == t：
      - 找到终点，从 t 开始沿父节点指针回溯，输出最终路径，结束
   c. 将 current 加入 ClosedList（表示该点已处理）
```

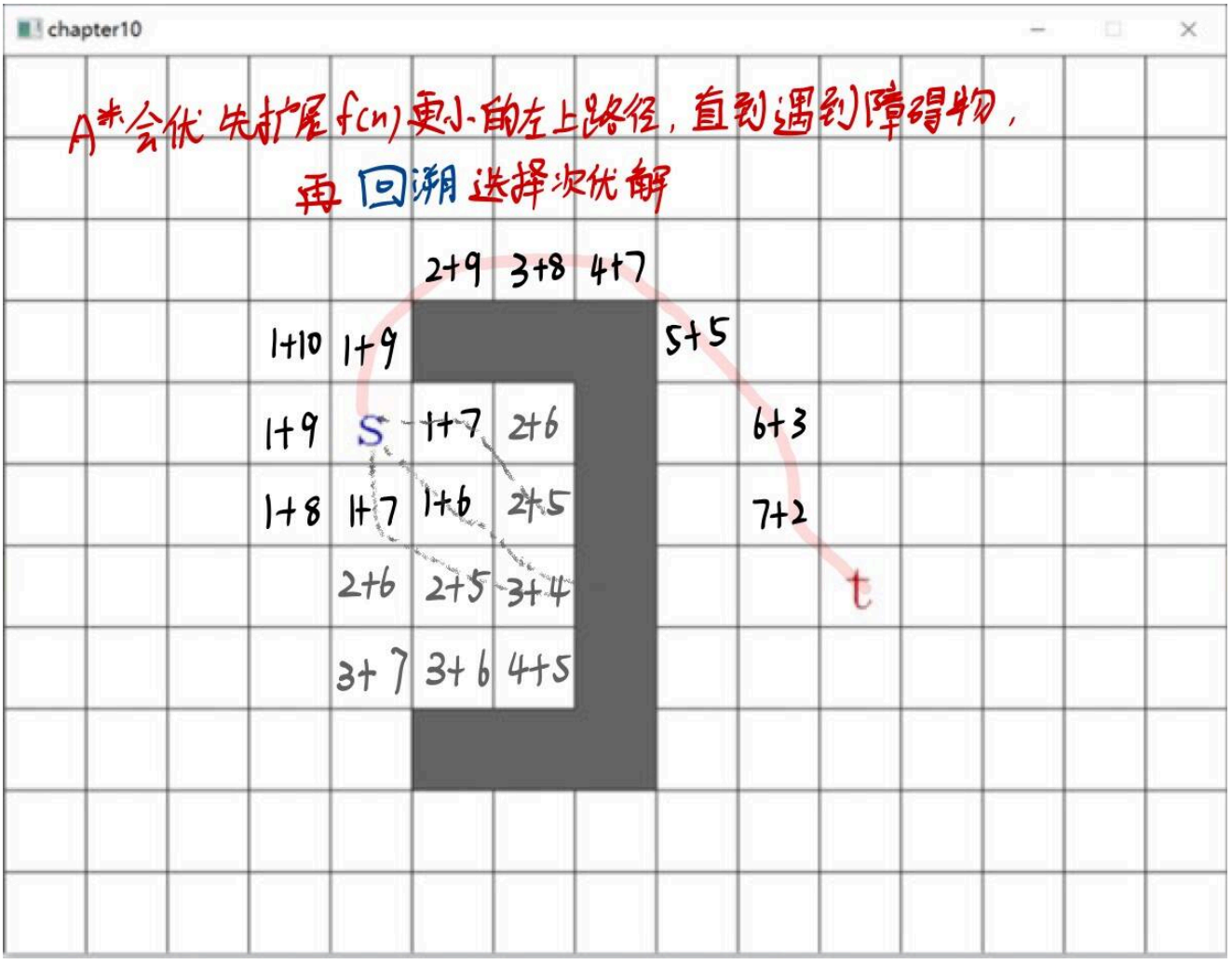
d. 遍历 current 的每一个相邻节点 neighbor（上、下、左、右）：
- 如果 neighbor 是障碍物 或 已经在 ClosedList 中：
  - 跳过，继续处理下一个邻居

- 计算从起点经过 current 到达 neighbor 的新代价：
  new_g = g(current) + 移动代价(通常为1)

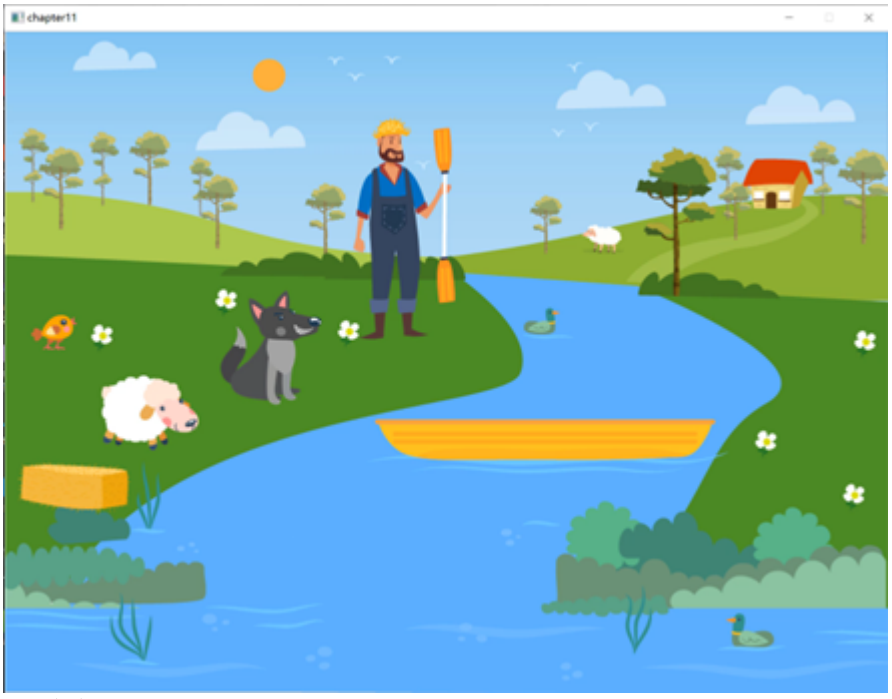
- 如果 neighbor 不在 OpenList 中：
  - 设置 neighbor 的父节点为 current
  - 计算 g(neighbor) = new_g
  - 计算 h(neighbor)（如使用曼哈顿距离或欧几里得距离）
  - 计算 f(neighbor) = g(neighbor) + h(neighbor)
  - 将 neighbor 放入 OpenList
- 否则（如果 neighbor 已在 OpenList 中）：
  - 如果 new_g < g(neighbor)：
    - 说明找到了更短的到达该点的路径，更新 neighbor 的父节点为 current
    - 更新 g(neighbor) = new_g，并重新计算 f(neighbor)

3. 如果 OpenList 为空且未找到终点：
- 路径规划失败（起点与终点不连通）

** (2)算法思想的示意图



2.农夫带着狼、羊、草过河，小船一次只能运载农夫和一样货物（也可以只运农夫），狼会吃羊、羊会吃草，只有农夫在时才安全。尝试设计算法求解操作步骤，让农夫和所有货物顺利过河。**试着描述分析过程和写出算法伪代码，并写出正确的操作步骤**。




**（1） 分析过程：状态空间建模

为了用算法求解，我们首先需要对问题进行形式化定义：

- **状态表示**：我们可以用一个四元组 $(F, W, S, C)$ 表示当前状态，其中 $F$ 代表农夫（Farmer）、$W$ 代表狼（Wolf）、$S$ 代表羊（Sheep）、$C$ 代表草（Cabbage）。
  - 每个变量取值可为 $0$（起始岸）或 $1$（目标岸）。
  - 初始状态：$(0, 0, 0, 0)$
  - 目标状态：$(1, 1, 1, 1)$
- **约束条件（非法状态）**：
  1. 如果农夫不在场（$F \neq W$ 且 $F \neq S$），狼会吃羊：即当 $W = S$ 且 $F \neq W$ 时，状态非法。
  2. 如果农夫不在场（$F \neq S$ 且 $F \neq C$），羊会吃草：即当 $S = C$ 且 $F \neq S$ 时，状态非法。
- **转移规则**：
  - 农夫每次过河必须划船，船上可以只载农夫，或者载农夫和一样货物。
  - 这意味着 $F$ 的值在每次移动后都会改变（$0 \to 1$ 或 $1 \to 0$）。
  - 如果有货物随行，该货物的取值随 $F$ 同步改变。

---

**（2） 算法伪代码 (基于广度优先搜索 BFS)

为了找到最少步数的解，使用 **BFS** 是最合适的策略。

Plaintext

```
算法 RiverCrossingSolver:
    // 定义状态：用四元组 (农夫, 狼, 羊, 草) 表示
    // 0 表示在起始岸，1 表示在目标岸
    START_STATE = (0, 0, 0, 0)
    TARGET_STATE = (1, 1, 1, 1)

    // 初始化工具
    创建 队列 Q                                  // 用于 BFS 搜索
    创建 集合 VISITED                            // 记录已访问过的状态，防止死循环
    创建 字典 PREV_MAP                           // 用于回溯查找最终的操作步骤路径

    将 START_STATE 入队 Q
    将 START_STATE 标记为已访问 VISITED

    while (队列 Q 不为空):
        当前状态 = Q.出队()

        // 成功条件：所有物品都已到达目标岸
        if (当前状态 == TARGET_STATE):
            返回 生成路径(PREV_MAP)

        // 尝试所有可能的移动方案
        // 方案包括：农夫带一样货物(狼/羊/草)，或者农夫自己划船
        for 移动对象 in [狼, 羊, 草, 无]:

            // 只有当货物和农夫在同一边时，农夫才能带走它
            if (移动对象 != 无 且 农夫位置 != 移动对象位置):
                继续下一次循环 (Skip)

            // 计算移动后的新状态
            下一状态 = 执行移动(当前状态, 移动对象)

            // 核心判断：新状态是否合法且未被访问过
            if (状态合法(下一状态) 且 下一状态 不在 VISITED 中):
                将 下一状态 标记为已访问 VISITED
                将 (当前状态 -> 下一状态) 的关系存入 PREV_MAP
                将 下一状态 入队 Q

函数 状态合法(state):
    (F, W, S, C) = state

    // 规则 1：农夫不在场时，狼和羊在一起 -> 非法
    if (W == S 且 F != W):
        return False

    // 规则 2：农夫不在场时，羊和草在一起 -> 非法
    if (S == C 且 F != S):
        return False
```

```
    // 其他情况均合法
    return True
```

---

**（3） 正确的操作步骤

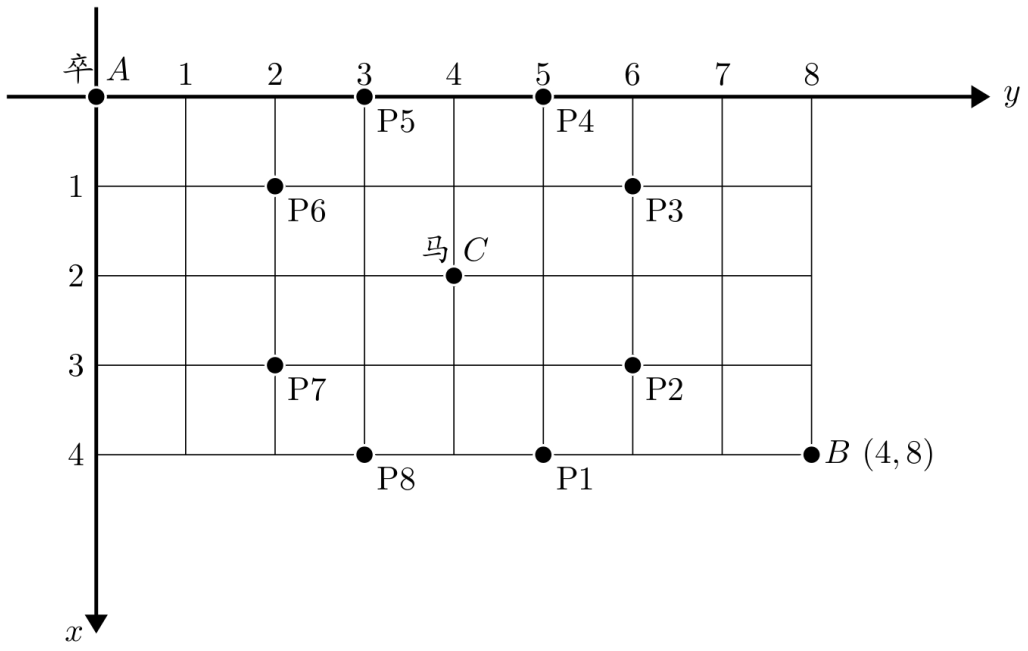
通过上述分析，可以得出最优解（共 7 步）：

| 步骤 | 动作 | 起始岸 (左) | 目标岸 (右) | 备注 |
|---|---|---|---|---|
| **0** | 初始状态 | 农, 狼, 羊, 草 | - | - |
| **1** | 农夫带**羊**过河 | 狼, 草 | 农, 羊 | 狼草安全 |
| **2** | 农夫独自返回 | 农, 狼, 草 | 羊 | - |
| **3** | 农夫带**狼**过河 | 草 | 农, 狼, 羊 | 农夫在，狼不吃羊 |
| **4** | 农夫带**羊**返回 | 农, 羊, 草 | 狼 | 防止羊吃草或狼吃羊 |
| **5** | 农夫带**草**过河 | 羊 | 农, 狼, 草 | 狼草安全 |
| **6** | 农夫独自返回 | 农, 羊 | 狼, 草 | - |
| **7** | 农夫带**羊**过河 | - | 农, 狼, 羊, 草 | **完成** |

> **注意**：第 3 步和第 5 步可以互换（即先带草过去，再带狼过去），逻辑依然成立。

**二、算法开发与编程求解。**分析问题，写出解题思路和算法伪代码；上机编程开发，写出对应的正确输出结果（每题20分，共40分，题型调整为选择题)**

**1.题目描述**
棋盘上 $A$ 点有一个过河卒，需要走到目标 $B$ 点。卒行走的规则：可以向下、或者向右。同时在棋盘上 $C$ 点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。棋盘用坐标表示，$A$ 点 $(0,0)$、$B$ 点 $(n,m)$，同样马的位置坐标是需要给出的。




现在要求你计算出卒从 $A$ 点能够到达 $B$ 点的路径的条数，假设马的位置是固定不动的，并不是卒走一步马走一步。
假设棋盘的大小为n=7行，m=9列，马的位置在(3,4)。过河卒从左上角(0,0)走到右下角(7,9)，编程求解出一共有**_215____**条不同的路径。

**递推式子：dp[i][j] = dp[i-1][j] + dp[i][j-1]

```
#include <bits/stdc++.h>
using namespace std;

long long dp[25][25];
bool ban[25][25];

int dx[9] = {0, 1, 1, -1, -1, 2, 2, -2, -2};
int dy[9] = {0, 2, -2, 2, -2, 1, -1, 1, -1};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m, x, y;
    cin >> n >> m >> x >> y;

    // 标记马及控制点
    for (int k = 0; k < 9; k++) {
        int nx = x + dx[k];
        int ny = y + dy[k];

        if (nx >= 0 && nx <= n && ny >= 0 && ny <= m) {
            ban[nx][ny] = true;
```

```
        }
    }

    dp[0][0] = 1;

    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {

            if (ban[i][j]) continue;
            if (i == 0 && j == 0) continue;

            if (i > 0) dp[i][j] += dp[i - 1][j];
            if (j > 0) dp[i][j] += dp[i][j - 1];
        }
    }

    cout << dp[n][m];

    return 0;
}
```

**本质上考递推，可以参考我洛谷4_1题2

**2.题目描述

**呵呵，有一天我做了一个梦，梦见了一种很奇怪的电梯。大楼的每一层楼都可以停电梯，而且第 i 层楼（1≤i≤N）上有一个数字 Ki（0≤Ki≤N）。电梯只有四个按钮：开，关，上，下。上下的层数等于当前楼层上的那个数字。当然，如果不能满足要求，相应的按钮就会失灵。例如：3,3,1,2,5 代表了 Ki（K1=3，K2=3，……），从 1 楼开始。
在 1 楼，按“上”可以到 4 楼，按“下”是不起作用的，因为没有 -2 楼。
那么，从 A 楼到 B 楼至少要按几次按钮呢?**

## 输入格式

共二行。

第一行为三个用空格隔开的正整数，表示 $N, A, B$（$1 \le N \le 200$，$1 \le A, B \le N$）。

第二行为 $N$ 个用空格隔开的非负整数，表示 $K_i$。

## 输出格式

一行，即最少按键次数，若无法到达，则输出 `-1`。

## 输入输出样例

输入 #1 复制

```
5 1 5
3 3 1 2 5
```

输出 #1 复制

```
3
```

## 说明/提示

对于 $100\%$ 的数据，$1 \le N \le 200$，$1 \le A, B \le N$，$0 \le K_i \le N$。

**请写出利用广度优先搜索求解该问题的思路和算法伪代码。**
**假设输入：**

**15 2 13**

**4 1 2 2 6 3 5 1 4 2 3 2 4 7 1**

请编程计算输出结果为：_6____

**详细见我博客洛谷9_1题解

思路：把每一层楼看成一个结点。
在第 i 层楼时，可以到达：

- `i + K[i]`（向上）
- `i - K[i]`（向下）
  每按一次按钮算走一步，所以题目就是求 **从 A 到 B 的最少步数**，这正是 **广度优先搜索 BFS** 的典型应用。

# 算法过程

1. 从 A 楼开始入队。
2. 每次取出当前楼层 x。
3. 尝试走到：

- `x + K[x]`
- `x - K[x]`

4. 如果楼层合法（1~N）且没访问过，则入队。
5. 当第一次到达 B 楼时，当前步数就是最少次数。
6. 若搜索结束仍没到达，输出 `-1`。

```
#include <iostream>
#include <queue>
#include <cstring>
using namespace std;

int n, A, B;
int k[205];
int dist[205];
bool vis[205];

int bfs() {
    queue<int> q;

    memset(dist, -1, sizeof(dist));
    q.push(A);
    dist[A] = 0;
    vis[A] = true;

    while (!q.empty()) {
        int cur = q.front();
        q.pop();

        if (cur == B) return dist[cur];

        // 向上
        int up = cur + k[cur];
        if (up >= 1 && up <= n && !vis[up]) {
            vis[up] = true;
            dist[up] = dist[cur] + 1;
            q.push(up);
        }

        // 向下
        int down = cur - k[cur];
        if (down >= 1 && down <= n && !vis[down]) {
            vis[down] = true;
            dist[down] = dist[cur] + 1;
            q.push(down);
        }
    }

    return -1;
```

```
}

int main() {
    cin >> n >> A >> B;

    for (int i = 1; i <= n; i++) {
        cin >> k[i];
    }

    cout << bfs();

    return 0;
}
```